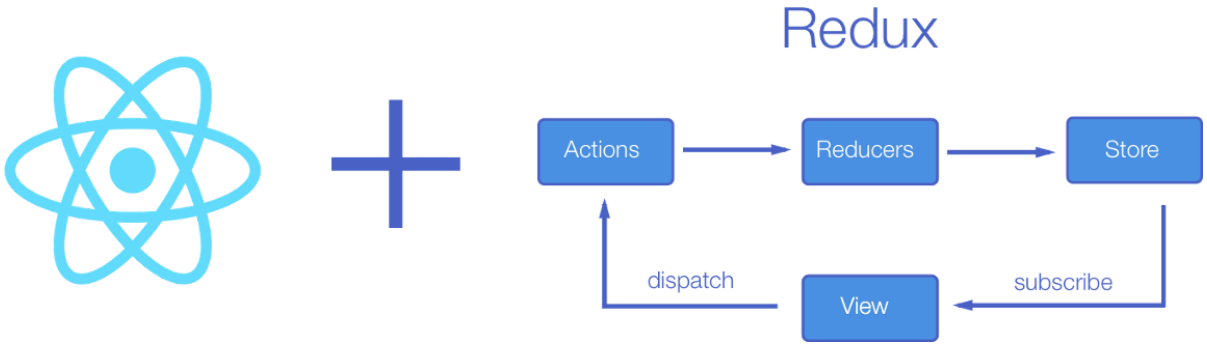
Date=25/11/2020
Lecture By=Manish Mahant
Subject ⇒Redux

IN PREVIOUS LECTURE **(QUICK RECAP) Date-24/11/2020**	**In Today's Lecture (Overview)**
React Lifecycle Lifecycle of Components Mounting Unmounting	Redux · An Introduction How Is It Different From MVC And Flux? Benefits Of Redux Functional Programming Where Can Redux Be Used? Building Parts Of Redux

Redux · An Introduction



Redux was created by Dan Abramov around June 2015. It was inspired by Facebook's Flux and functional programming language Elm. Redux got popular very quickly because of its simplicity, small size (only 2 KB) and great documentation. If you want to learn how Redux works internally and dive deep into the library

Redux is used mostly for application state management. To summarize it, Redux maintains the state of an entire application in a single immutable state tree (object), which can't be changed directly. When something changes, a new

object is created (using actions and reducers). We'll go over the core concepts in detail below.

How Is It Different From MVC And Flux?

To give some perspective, let's take the classic model-view-controller (MVC) pattern, since most developers are familiar with it. In MVC architecture, there is a clear separation between data (model), presentation (view) and logic (controller). There is one issue with this, especially in large-scale applications: The flow of data is bidirectional. This means that one change (a user input or API response) can affect the state of an application in many places in the code — for example, two-way data binding. That can be hard to maintain and debug.
Flux is very similar to Redux. The main difference is that Flux has multiple stores that change the state of the application, and it broadcasts these changes as events. Components can subscribe to these events to sync with the current state. Redux doesn't have a dispatcher, which in Flux is used to broadcast payloads to registered callbacks. Another difference in Flux is that many varieties are available, and that creates some confusion and inconsistency.

Benefits Of Redux

You may be asking, "Why would I need to use Redux?" Great question. There are a few benefits of using Redux in your next application:

- Predictability of outcome
  There is always one source of truth, the store, with no confusion about how to sync the current state with actions and other parts of the application.
- Maintainability
  Having a predictable outcome and strict structure makes the code easier to maintain.

- Organization
  Redux is stricter about how code should be organized, which makes code more consistent and easier for a team to work with.
- Server rendering
  This is very useful, especially for the initial render, making for a better user experience or search engine optimization. Just pass the store created on the server to the client side.
- Developer tools
  Developers can track everything going on in the app in real time, from actions to state changes.
- Community and ecosystem
  This is a huge plus whenever you're learning or using any library or framework. Having a community behind Redux makes it even more appealing to use.
- Ease of testing
  The first rule of writing testable code is to write small functions that do only one thing and that are independent. Redux's code is mostly functions that are just that: small, pure and isolated.

Functional Programming

As mentioned, Redux was built on top of functional programming concepts. Understanding these concepts is very important to understanding how and why Redux works the way it does. Let's review the fundamental concepts of functional programming:

- It is able to treat functions as first-class objects.
- It is able to pass functions as arguments.
- It is able to control flow using functions, recursions and arrays.
- It is able to use pure, recursive, higher-order, closure and anonymous functions.
- It is able to use helper functions, such as map, filter and reduce.
- It is able to chain functions together.
- The state doesn't change (i.e. it's immutable).

- The order of code execution is not important.

Functional programming allows us to write cleaner and more modular code. By writing smaller and simpler functions that are isolated in scope and logic, we can make code much easier to test, maintain and debug. Now these smaller functions become reusable code, and that allows you to write less code, and less code is a good thing. The functions can be copied and pasted anywhere without any modification. Functions that are isolated in scope and that perform only one task will depend less on other modules in an app, and this reduced coupling is another benefit of functional programming.

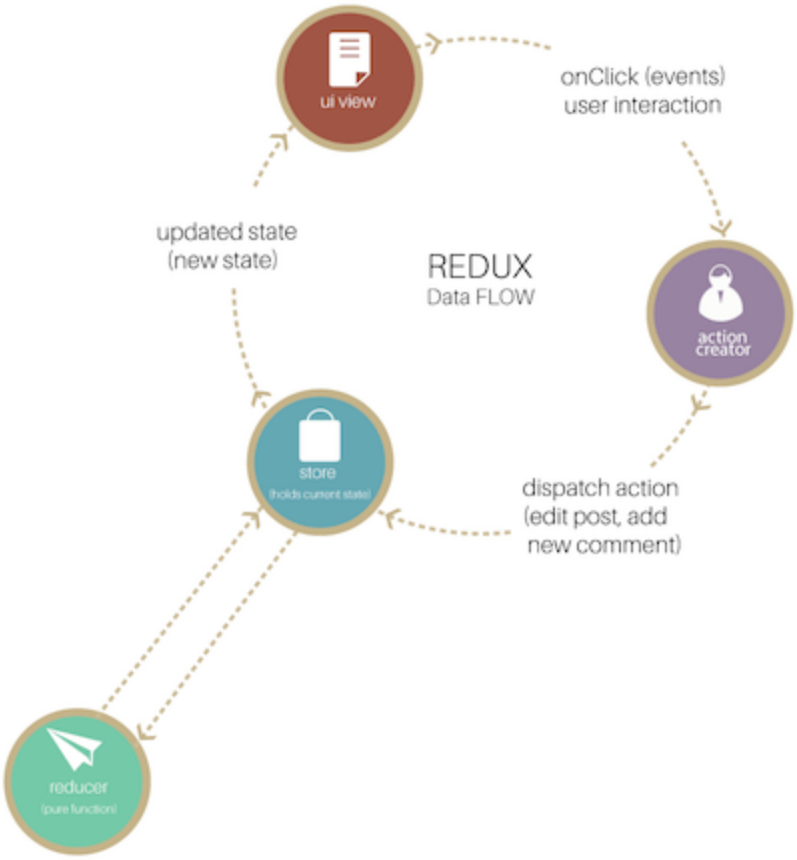
Where Can Redux Be Used?

Most developers associate Redux with React, but it can be used with any other view library. For instance, you can use Redux with AngularJS, Vue.js, Polymer, Ember, Backbone.js and Meteor. Redux plus React, though, is still the most common combination. Make sure to learn React in the right order: The best guide is Pete Hunt's, which is very helpful for developers who are getting started with React and are overwhelmed with everything going on in the ecosystem. JavaScript fatigue is a legitimate concern among front-end developers, both new or experienced, so take the time to learn React or Redux the right way in the right order.

One of the reasons Redux is awesome is its ecosystem. So many articles, tutorials, middleware, tools and boilerplates are available. Personally, I use David Zukowski's boilerplate because it has everything one needs to build a JavaScript application, with React, Redux and React Router.

Building Parts Of Redux

Redux concepts might sound complicated or fancy, but they're simple. Remember that the library is only 2 KB. Redux has three building parts: actions, store and reducers.



ACTIONS

In a nutshell, actions are events. Actions send data from the application (user interactions, internal events such as API calls, and form submissions) to the store. The store gets information only from actions. Internal actions are simple JavaScript objects that have a `type` property (usually constant), describing the type of action and payload of information being sent to the store.

```
{
    type: LOGIN_FORM_SUBMIT,
    payload: {username: 'alex', password: '123456'}
```

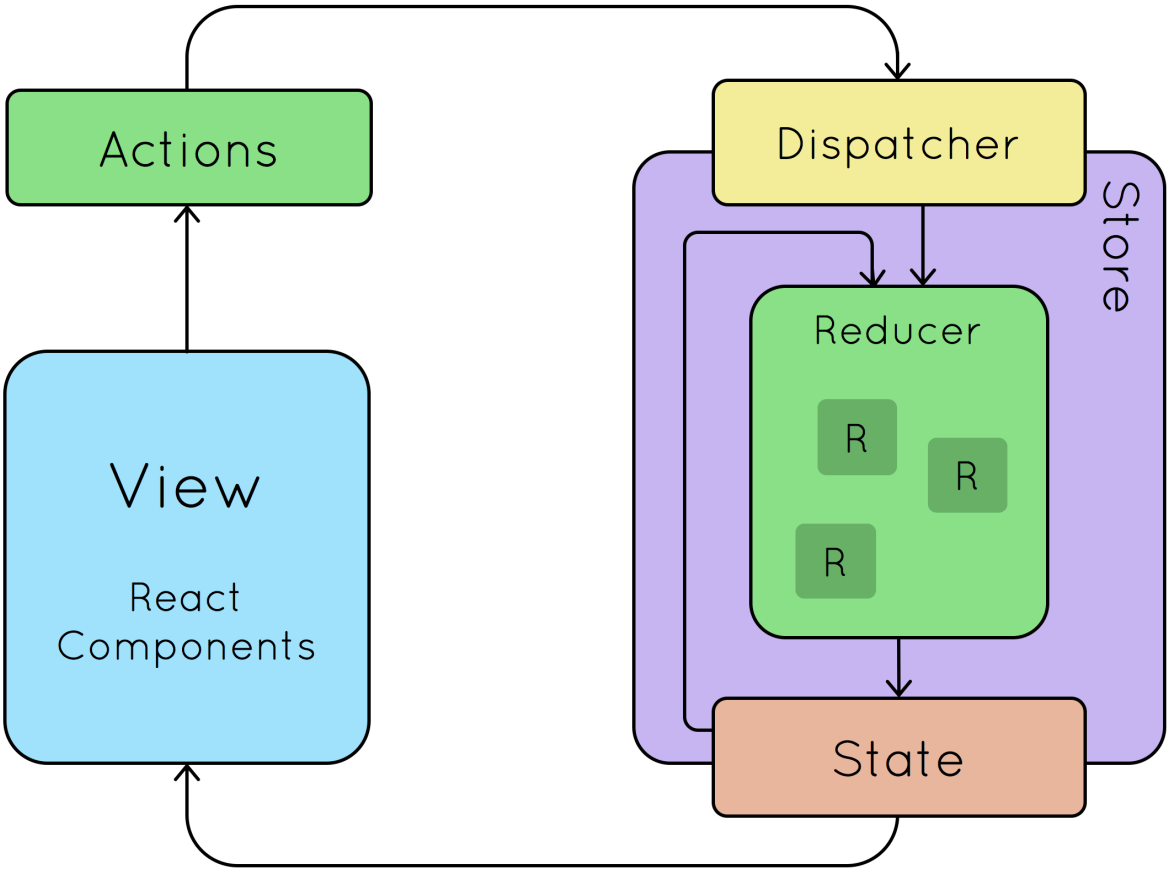
```
}
```

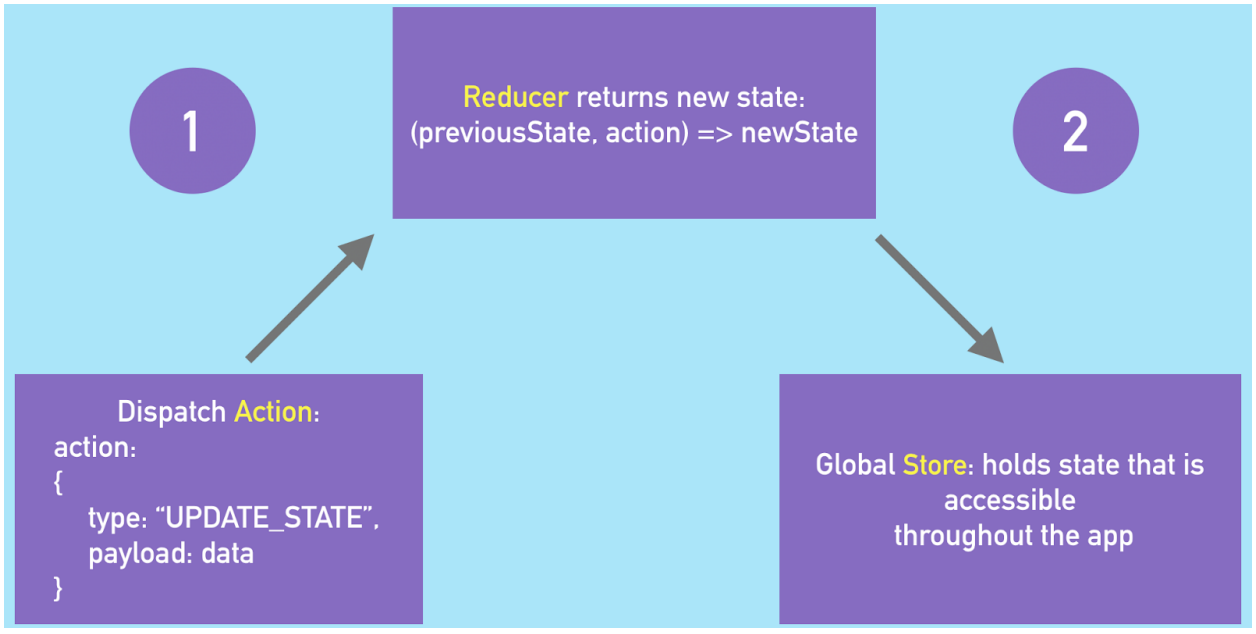
Actions are created with action creators. That sounds obvious, I know. They are just functions that return actions.

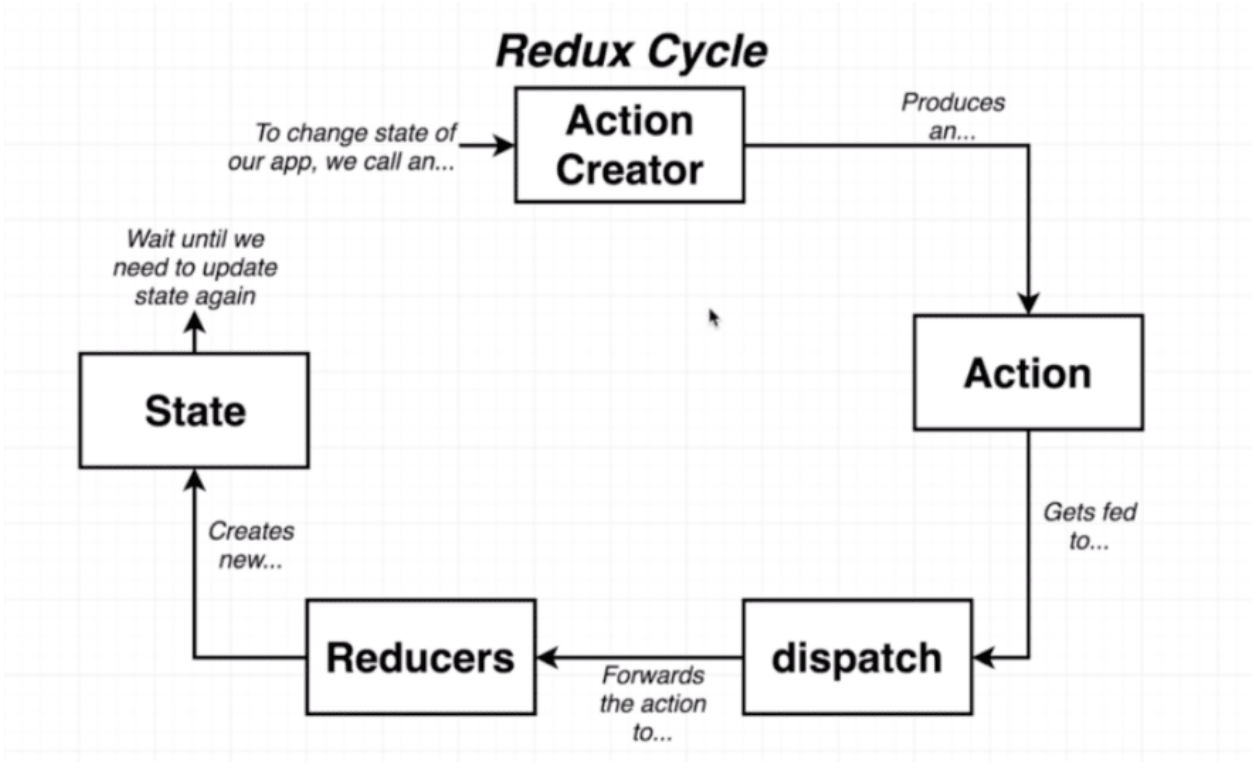
```
function authUser(form) {
    return {
        type: LOGIN_FORM_SUBMIT,
        payload: form
    }
}
```

Calling actions anywhere in the app, then, is very easy. Use the `dispatch` method, like so:

```
dispatch(authUser(form));
```


Actions
Dispatcher
Store
Reducer
R
R
R
View
React
Components
State


1
Reducer returns new state:
(previousState, action) => newState
2
Dispatch Action:
action:
{
type: "UPDATE_STATE",
payload: data
}
Global Store: holds state that is
accessible
throughout the app


Redux Cycle
To change state of our app, we call an...
Action Creator
Produces an...
Action
Gets fed to...
dispatch
Forwards the action to...
Reducers
Creates new...
State
Wait until we need to update state again